CamJam! Workshop: Node-RED and getting started on the Internet of Things

http://nodered.org

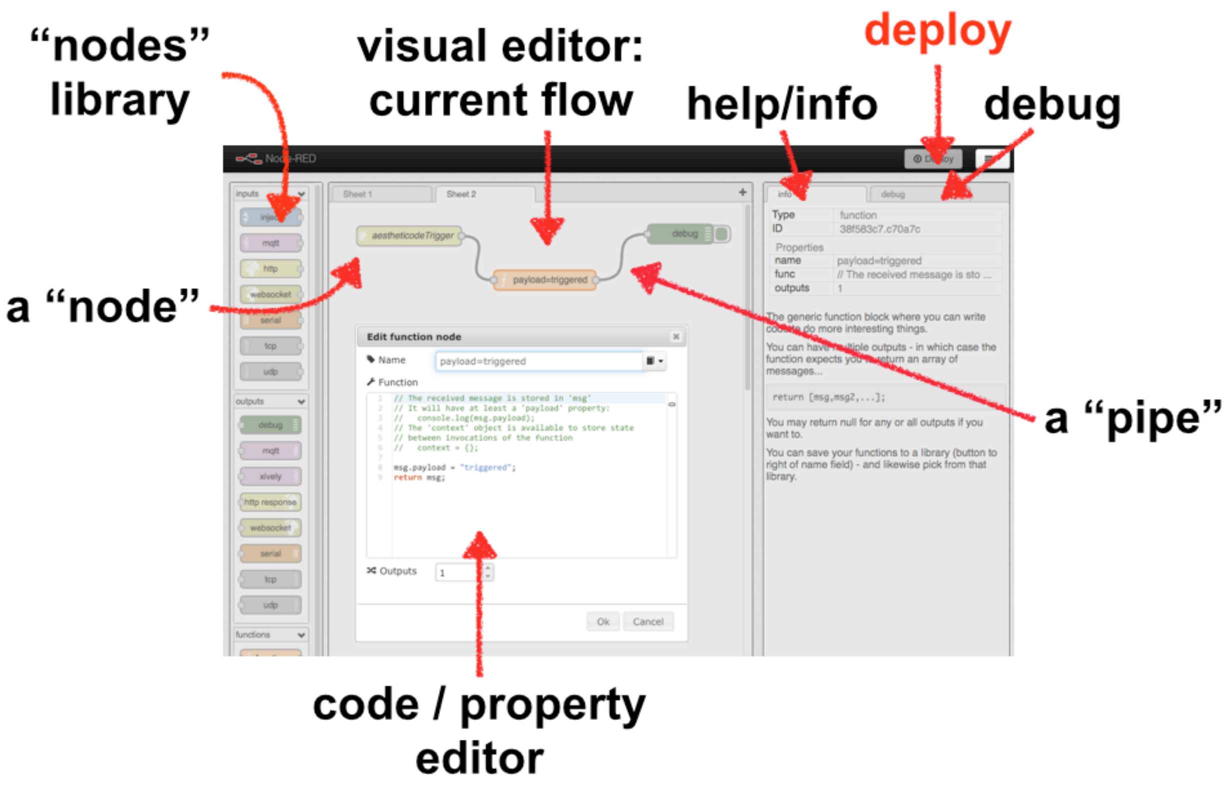
Tinamous .com

http://shop.ciseco.co.uk

- **Node-RED is a visual tool for wiring the Internet of Things (IoT).** Node-RED is platform-independent, but has been developed with small computers such as the Raspberry Pi in mind.

- Traditional IoT development can be very technical: Access to the GPIO and other hardware requires skills in C or assembler, output of data to web services or sending tweets and emails requires the use of complex APIs. **Node-RED takes care of the technicalities and lets you concentrate on the logic of your workflow.**

- While most programming in Node-RED is done visually using pre-defined functions ("nodes"), any **additional functionality can be added in JavaScript.**



Name: payload=triggered

Function:

```
// The received message is stored in 'msg'
// It will have at least a 'payload' property:
//   console.log(msg.payload);
// The 'context' object is available to store state
// between invocations of the function
//   context = {};

msg.payload = "triggered";
return msg;
```

Outputs: 1

Type	function
ID	38f583c7.c70a7c
Properties	
name	payload=triggered
func	// The received message is sto ...
outputs	1

The generic function block where you can write code to do more interesting things.

You can have multiple outputs - in which case the function expects you to return an array of messages...

```
return [msg,msg2,...];
```

You may return null for any or all outputs if you want to.

You can save your functions to a library (button to right of name field) - and likewise pick from that library.

WORKSHOP CONTENT: In this workshop, we're going to use Node-RED to interact with a Ciseco **Slice of Radio.** This is a low-cost radio shield that enables the Raspberry Pi to receive messages from **wireless sensors** (such as temperature or light level) and send messages to **change wireless actuators** (such as a relay or light). We will also discuss how to send sensor data to IoT platforms on the Internet.

CamJam! Workshop: Node-RED and getting started on the Internet of Things

Technical background: For this workshop, you will find a Raspberry Pi with a Slice of Radio and Node-RED already installed. While the installation of both hardware and software is relatively easy, it would be difficult to include this step within the time constraints of the exercise. For completeness, this is what's happened to a fresh and up-to-date Raspian installation on your SD card:

1. Remove any system I/O through the serial port (*/dev/ttyAMA0*) from */etc/inittab* and */boot/cmdline.txt*. See e.g. *http://openmicros.org/index.php/articles/94-ciseco-product-documentation/raspberry-pi/283-setting-up-my-raspberry-pi*

2. Install node.js and npm (e.g. by *sudo apt-get install nodejs npm*)

3. Clone Node-RED from Github and install as described here: *http://nodered.org/docs/getting-started/installation.html*

4. Install the node that encapsulates communication with the serial port by issuing: *sudo npm install serialport*

5. The default Midori browser has its quirks. We're using Chromium, installed by: *sudo apt-get install chromium*

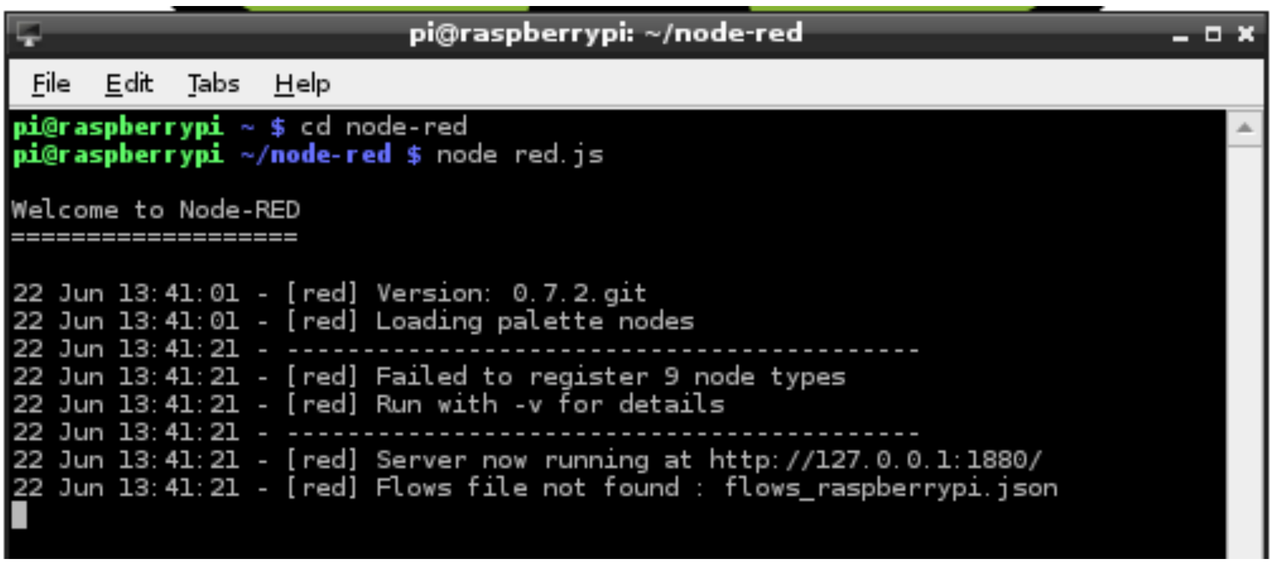
Note: To leverage the power of Node-RED, consider installing nodes for email or raw GPIO access as well. The latter is described here: *http://nodered.org/docs/hardware/raspberrypi.html*

1) Exercise: Starting Node-RED as Raspberry Pi user

Node-RED can be installed as a service on the Raspberry Pi, i.e. as a program that's always executed when your Pi is running. However, this is only useful if you want to commit your Pi for this particular use as it can consume considerable resources. For everyone else, it's recommended to start Node-RED only when needed:

1. Open the LXTerminal to see a console that allows you to enter Linux commands.
2. Change into the Node-RED directory by issuing "cd node-red".
3. Start Node-RED by typing "node red.js".

You should now see Node-RED starting up – that may take a few seconds:



```
pi@raspberrypi ~ $ cd node-red
pi@raspberrypi ~/node-red $ node red.js

Welcome to Node-RED
===================

22 Jun 13:41:01 - [red] Version: 0.7.2.git
22 Jun 13:41:01 - [red] Loading palette nodes
22 Jun 13:41:21 - ------------------------------------------
22 Jun 13:41:21 - [red] Failed to register 9 node types
22 Jun 13:41:21 - [red] Run with -v for details
22 Jun 13:41:21 - ------------------------------------------
22 Jun 13:41:21 - [red] Server now running at http://127.0.0.1:1880/
22 Jun 13:41:21 - [red] Flows file not found : flows_raspberrypi.json
```

Congratulations. You're now ready for the exercises.

Node-RED represents a server on the basis of node.js and interacts with the user through a graphical user interface. It can be reached on port 1880. **To use Node-RED, open a web browser and direct it to http://localhost:1880**

It's useful to remember that Node-RED acts as a server in your entire network. That is, if your Raspberry Pi's internal IP address is something like 192.x.x.x, every computer in your network can open the Node-RED GUI through http://192.x.x.x:1880. You can make your system more restricted/secure by following the configuration advice on *http://nodered.org/docs/configuration.html.*

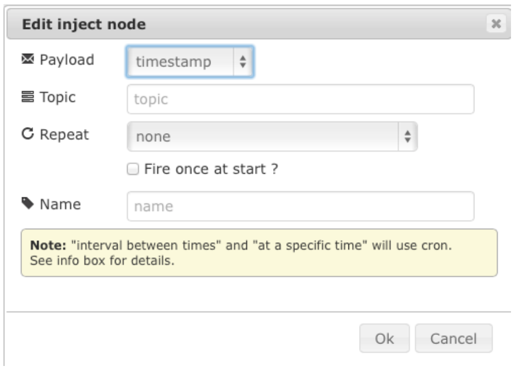
2) Exercise: Your first flow

The best way to explain "a flow" is by creating one. In this mini flow, we're going to inject a value into our debug window (refer to page 1 for what the GUI elements are called).

1. Open the **Chromium Web Browser**. It supports JavaScript better than the default Midori browser.
2. In the address line, enter **localhost:1880**. You will then see the Node-RED GUI.
3. Drag and drop an **"inject" node** from the nodes library into the flow editor (once you've chosen the inject node, you should see some general explanation about its functionality in the info pane – no need to read that now).
4. Drag and drop a **"debug" node** from the nodes library into the flow editor.
5. Create a pipe between the inject and debug nodes by **drawing a connection** between their small grey rounded rectangles.
6. Change from the info pane to the **debug pane** (upper right).
7. **Deploy** (=start) your flow.
8. Once deployed, press the left blue rectangle that's attached to the inject node. Check what's happening in the debug pane.

3) Exercise: Topics and payloads

While it is possible to push complicated data structures through pipes in Node-RED, the default is a (topic / payload) tuple, which could be interpreted as subject and body of an email.

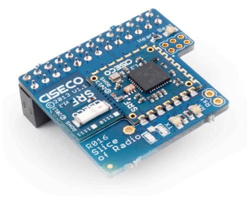

Edit inject node

Payload: timestamp
Topic: topic
Repeat: none
Fire once at start ?
Name: name

Note: "interval between times" and "at a specific time" will use cron. See info box for details.

1. Double-click your inject node.
2. Change the **properties of your inject node** so that it sends a *string* "the message" as payload and "the envelope" as topic.
3. **Deploy your flow and check** for the outcome in the debug window.

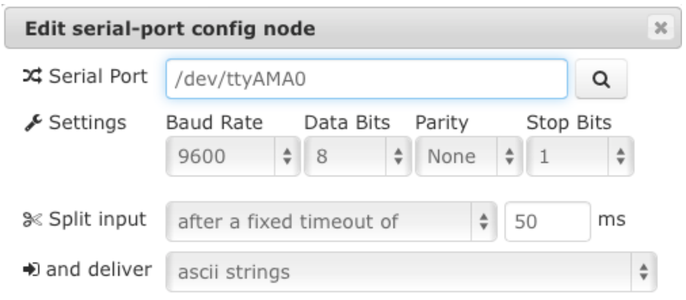
4) Exercise: Receiving data from the serial port



Triggering a flow through the inject node is obviously of limited utility. **Let's interact with the real world!** The Slice of Radio is a small module that enables the Raspberry Pi to send and receive radio frequency messages. Once connected to the GPIO, the radio communicates with the serial (tx, rx) pins.

In our workshop, you will receive wireless radio messages from a temperature sensor. In Node-RED, the **"serial" node** facilitates serial communication with the sensor.

1. Drag and drop a serial node into the flow editor. You can call it "radio".
2. **Connect the serial node to your debug** node you've used in the last script.
3. **Configure the serial node** to talk to serial port */dev/ttyAMA0* (the standard Linux name of the serial port on the Raspberry Pi), expect 9600 baud 8N1 communication ("how a character is encoded") and split your input after 50 milliseconds.



Edit serial-port config node

Serial Port: /dev/ttyAMA0

Settings: Baud Rate 9600, Data Bits 8, Parity None, Stop Bits 1

Split input: after a fixed timeout of 50 ms

and deliver: ascii strings

4. **Deploy your flow** and wait for your first radio message (up to 30 sec).

5) Exercise: Use JavaScript to extract what's useful from LLAP messages

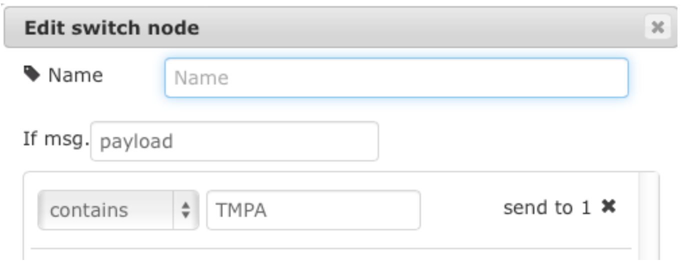
Take a deep breath. Now you're going learn exciting technical stuff and JavaScript!

A radio has no notion of the physical beginning or end of a transmission. However, we know that our **Ciseco temperature sensor** speaks **LLAP**, or *Lightweight Logical Application Protocol*[1]. Every LLAP sentence is 12 characters long. That means, unless there is a lot of radio chatter about, every burst of 12 characters should be a complete LLAP message. An exemplary message we're looking out for is following the convention **aT1TMPA23.5-**, meaning the message has started ("a"), it comes from sensor T1, and the temperature ("TMPA") is 23.5, plus a spacer ("-"): 12 bytes. While Ciseco radios can send any length of information (yes, even the transcript of Miles' recent speech at the IoT 14 Meeting), for their own sensor line they've chosen to encode the output in LLAP.

[1] http://shop.ciseco.co.uk/llap/ & http://openmicros.org/index.php/articles/85-llap-lightweight-local-automation-protocol/297

In this workshop, we are going to store our most recent temperature readings to a platform called Tinamous (https://tinamous.com). There are also other platforms, all for slightly different use cases[2]. A common format for sensor data transmission to these platforms is MQTT, which follows a topic / payload convention.

1. Drag and drop a **"switch" node** into the flow editor. Configure it in a way that only messages with a TMPA statement get forwarded and other messages are disregarded:



Edit switch node

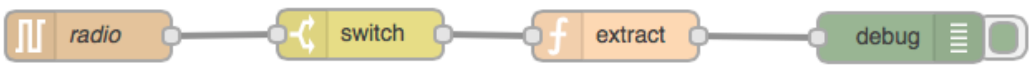
Name: Name

If msg. payload

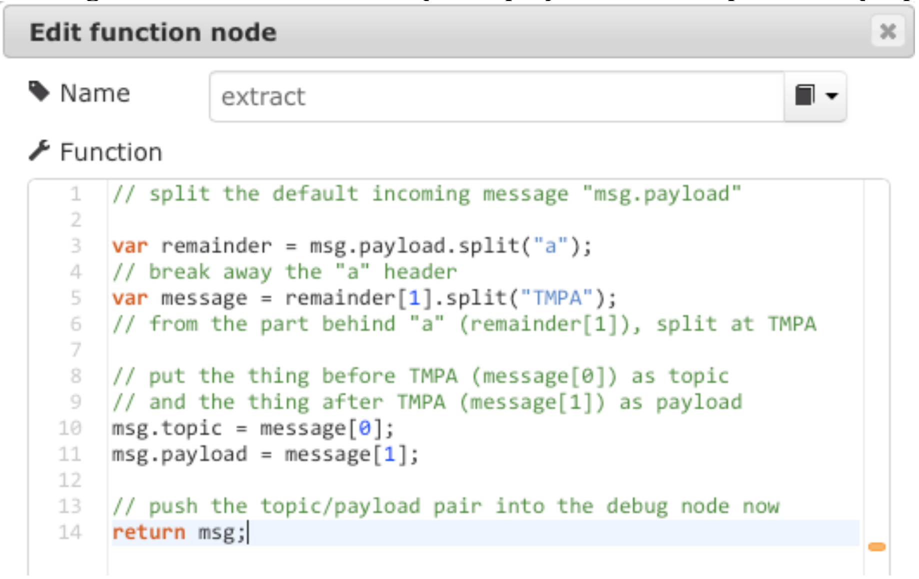
contains TMPA send to 1

2. Drag and drop a **"function" node** into the flow editor.

3. **Connect** the switch node to your radio, and the function node to the switch node so that it sits in between the switch and the debug node.



4. Open the function node and **write some JavaScript code** that separates the LLAP message into the sensor name (for topic) and the temperature (as payload).



Edit function node

Name: extract

Function

```
// split the default incoming message "msg.payload"

var remainder = msg.payload.split("a");
// break away the "a" header
var message = remainder[1].split("TMPA");
// from the part behind "a" (remainder[1]), split at TMPA

// put the thing before TMPA (message[0]) as topic
// and the thing after TMPA (message[1]) as payload
msg.topic = message[0];
msg.payload = message[1];

// push the topic/payload pair into the debug node now
return msg;
```

(Note: You don't have to type all the green things. These are just comments to explain a little bit what's going on in the code. Concentrate on the things that are not green.)

5. **Deploy your workflow** and check what's going on in your debug pane.

[2] See http://logic.sysbiol.cam.ac.uk/?p=1473 for a review for some working with Node-RED

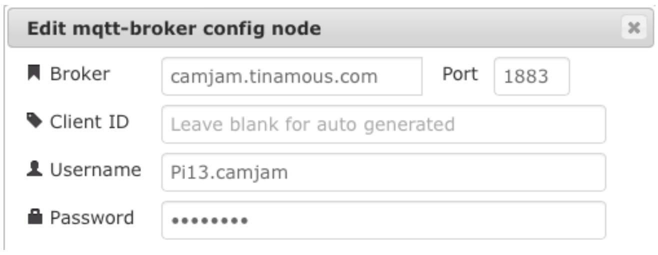
Comments and further experiments:

msg.payload and msg.topic are available at the start of every function node!

You may, by now, have realised that there are not only TMPA messages, but also AWAKE, SLEEPING and BATT… If you're a keen JavaScript programmer, write some code that avoids the switch node and does all the filtering in the function node.

6) Exercise: Connect to Tinamous through MQTT

1. Drag and drop a **"MQTT" output node** into the flow editor.
2. **Connect** the MQTT node to the *extract* function node.
3. **Configure** the MQTT node. We have previously set up a *camjam* account on Tinamous and defined "PiXX" devices (where XX is your table number; here: 13). These are the details you will need to enter in the configuration of your MQTT broker:

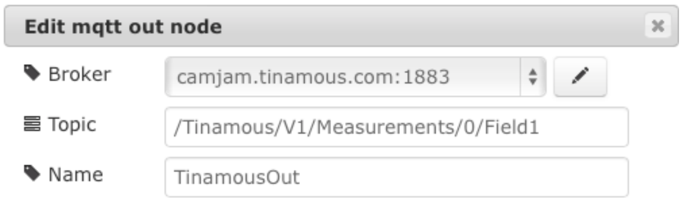


Edit mqtt-broker config node

Broker	camjam.tinamous.com	Port	1883
Client ID	Leave blank for auto generated		
Username	Pi13.camjam		
Password	••••••••		

(Our very creative password is camjam2camjam, username: PiXX "dot" camjam).

In the MQTT node itself you specificy as topic /Tinamous/V1/Measurements/0/Field1



Edit mqtt out node

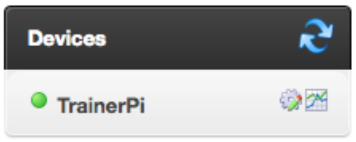
Broker	camjam.tinamous.com:1883
Topic	/Tinamous/V1/Measurements/0/Field1
Name	TinamousOut

(most of this is Tinamous nomenclature – look at their MQTT online help for details).

4. **Deploy your workflow** and visit *https://camjam.tinamous.com* in your web browser.

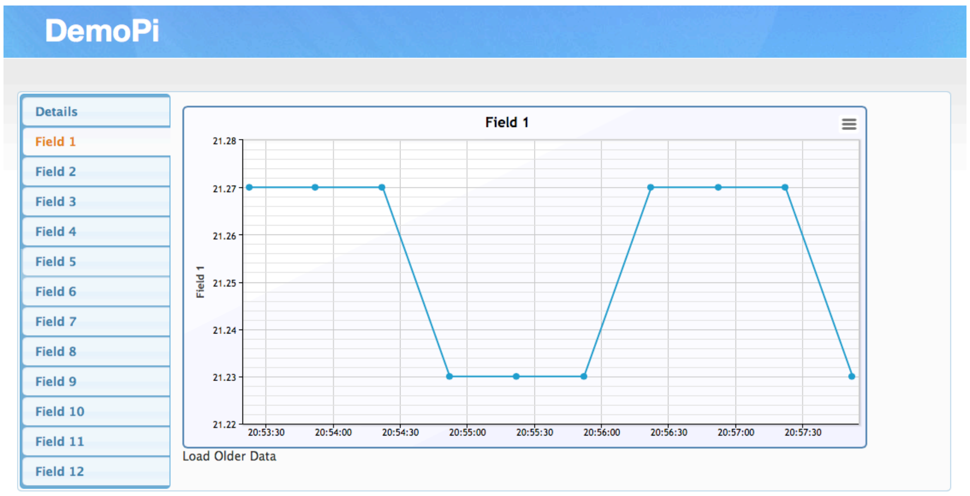
Quick Excursion: Viewing your data on Tinamous

1. Point your browser to https://camjam.tinamous.com (camjam / camjam2camjam).
2. Click on the small line chart icon next to your device (e.g. Pi13).



3. Click on the Field1 that we've designated as temperature.

4. Voila!



7) Exercise: Control stuff through Node-RED

In the next step, we're going to **control an actuator**. That's the smart word for a thing that does something in response. Actuators are: lights, relays, switches, etc. In our example, every Raspberry Pi in the workshop is going to have a number assigned. Your number translates into a position in an Adafruit NeoPixel Ring. The NeoPixel Ring is connected to a XinoRF wireless Arduino-clone (it's the same microcontroller that's part of the **Ciseco RasWik Wireless Inventor Kit**), which runs some code to listen to your radio messages and trigger a response.

We've set up the XinoRF so that it responds to your LLAP messages of the format **aRFttrrggbb-**, where tt is your table number (01..12), and rrggbb is the hexadecimal code of your desired colour. So FF0000 is "red", 00FF00 is "green" and 0000FF is "blue".

1. Drag an **outgoing "serial" node** into your project and **connect it to an inject node**.
2. **Configure** the inject node so that it injects a payload in aRFttrrggbb- format.
3. **Deploy your workflow** and try to identify your position in the NeoPixel Ring.

Would it not be wonderful to translate the temperature into a colour tone?

You can do it!

1. Write yourself some code (or use a switch node!) that sends a different colour code for different temperatures.

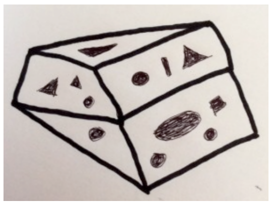
Note: As soon as many groups start to automate this step, we're going to have a dozen Raspberry Pis fire a radio message once the temperature sensor has triggered their flow. You can prevent the XinoRF from being overrun by inserting **a "delay" node** with a random delay (say, between 0 and 1000 milliseconds) before the outgoing radio node.

8) Additional Exercises… …in your own time

Node-RED is an incredibly powerful framework that allows you to do things in very little time. The **official directory of flows** donated to the community is here *http://flows.nodered.org* and they can easily be imported by copying & pasting the JSON-formatted code.

A few suggestions and examples that I have described over the past months are here:

- **Triggering Node-RED with drawings**: The Aestheticodes project uses a QR code like method to encode information in beautiful drawings. Draw a picture, take a photograph with your mobile and trigger the debug node doing that: *http://logic.sysbiol.cam.ac.uk/?p=1514*



- **Control Minecraft with Node-RED**: The Minecraft Pi Edition can be controlled through Python, but that may not be easily accessible for everyone. With a MQTT-to-Minecraft bridge, the Node-RED inject nodes can be used to control Steve: *http://logic.sysbiol.cam.ac.uk/?p=1499*

- Got an AirPi shield? **Monitor your room climate with AirPi and Node-RED**: *http://logic.sysbiol.cam.ac.uk/?p=1423*